Pr. Karima AISSAOUI

# TP/TD : Rappel du langage C : Tableaux

**Objectif :** Rappeler les notions fondamentales des tableaux en langage C

## Exercice 1

Soit le tableau suivant :

```
int[] tab = {12, 15, 13, 10, 8, 9, 13, 14};
```

Question 1

Ecrire un programme qui saisit un entier au clavier et qui recherche si cet entier appartient au tableau (réponse de type oui/non).

Question 2

Ecrire un programme qui saisit un entier au clavier et qui recherche si cet entier appartient au tableau donné au début de l'exercice. Au cas où la réponse est positive, l'indice de cet entier dans le tableau est affiché. S'il y a plusieurs occurrences, le dernier indice est affiché.

## Exercice 2

Ecrire un programme qui saisit un entier et un indice et met cet entier dans le tableau à cet indice. Il faudra vérifier que l'indice correspond bien à une case du tableau (par exemple 0 ou 3 sont corrects, mais -1 ou 100 ne sont pas des indices corrects pour ce tableau). Le programme affichera le contenu du tableau avant et après cette transformation.

## Exercice 3

Ecrire un programme qui saisit la dimension N d'un tableau de int (le tableau est initialement définit avec une taille maximum MAX que N ne doit pas excéder) remplit le tableau par des valeurs entrées au clavier et l'affiche. Le programme doit ensuite effacer toutes les occurrences de la valeur 0 dans le tableau, insérer les éléments restants et afficher le tableau ainsi modifié.

## Exercice 4

Ecrire un programme qui lit les dimensions L et C d'un tableau T à deux dimensions du type int (dimensions maximales: 50 lignes et 50 colonnes). Remplir le tableau par des valeurs entrées au clavier et afficher le tableau ainsi que la somme de tous ses éléments.



Scanned with CamScanner

Pr. Karima AISSAOUI

# TP/TD 2 : Pointeurs et allocation dynamique

**Objectif :** Maîtriser les notions d'adresse et de pointeur

## Exercice 1

- Déclarer un pointeur sur int et l'initialiser par le pointeur NULL.
- Déclarer un pointeur sur pointeur sur int.
- Déclarer une variable de type int.
- Initialiser le pointeur sur int par l'adresse de cette variable.
- Initialiser le pointeur sur pointeur sur int par l'adresse du pointeur sur int.
- Allouer dynamiquement un espace de mémoire suffisant pour contenir une variable de type int, et stocker son adresse dans le pointeur sur int.

## Exercice 2

- Déclarer deux pointeurs sur `float` .
- Déclarer deux variables de type `float` .
- Initialiser chacun des pointeurs avec les adresses des deux variables `float` .
- Affecter la valeur 12.5 à la première variable en utilisant son pointeur.
- Affecter la valeur 5.76 à la deuxième variable en utilisant son pointeur.
- Échanger le contenu des deux variables en utilisant leurs pointeurs.

## Exercice 3

- Écrire une procédure "permute" qui permet de permuter les valeurs de deux variables entières et écrire un programme dans lequel on saisira deux nombres entiers avant de faire appel a cette procédure et d'afficher le contenue de ces variable afin de vérifier la permutation.

## Exercice 4

- Utiliser l'allocation dynamique pour créer un tableau de 15 variables de type `float` . Initialiser ce tableau avec des zéros, puis libérer la mémoire allouée si l'allocation a été effectuée avec succès.

## Exercice 5

- Déclarer un double pointeur sur `float` , puis utiliser une fonction pour effectuer l'allocation dynamique d'un tableau de 15 variables de type `float`.
- Utiliser une fonction pour initialiser ce tableau avec des zéros si l'allocation a été effectuée avec succès.
- Puis libérer la mémoire allouée toujours si l'allocation a été effectuée avec succès.



Scanned with CamScanner

USMBA/ ENSAF
Structures de données

Pr. Karima AISSAOUI

# TP/TD 03 : Listes chaînées simples

**Objectif :** Maîtriser les manipulations des listes chaînées simples.

## Exercice 1

1. Créer la structure Element composée de :
   - Un entier valeur
   - Un pointeur vers l'élément suivant
2. Créer la structure List composée de :
   - Longueur de type « Short int »
   - Un élément constituant la tête de la liste
3. Initialiser votre liste avec un élément d'une valeur « int valeur »
4. Ajouter un élément en tête de la liste
5. Ajouter un élément en fin de liste
6. Afficher votre liste après chaque manipulation
7. Supprimer la tête de la liste
8. Supprimer la fin de la liste
9. Afficher votre liste finale



Scanned with CamScanner

USMBA/ ENSAF
Structures de données

Pr. Karima AISSAOUI

# TP/TD 04 : Listes doublement chaînées

**Objectif :** Maîtriser les manipulations des listes doublement chaînées.

## Exercice 1

1. Créer la structure *node* composée de :
    - Un entier *data*
    - Un pointeur de type *struct node* vers l'élément suivant
    - Un pointeur de type *struct node* vers l'élément suivant

2. Créer la structure dlist composée de :
    - Longueur de type « Short int »
    - Un pointeur de type *struct node* constituant la tête de la liste
    - Un pointeur de type *struct node* constituant la queue de la liste

3. Allouer de l'espace pour votre nouvelle liste.
4. Ajouter un *node* en fin de liste
5. Ajouter un *node* en tête de la liste
6. Ajouter un *node* selon une position demandée par l'utilisateur (Traiter les 3 cas possibles de cette position comme vu dans le cours)
7. Supprimer un *node* selon une valeur (la valeur est demandée par l'utilisateur)
8. Supprimer un *node* selon une position (la position est demandée par l'utilisateur)
9. Faites la recherche d'un *node* selon sa valeur (la valeur est demandée par l'utilisateur)
10. Faites la recherche d'un ensemble d'éléments selon une valeur (retourner une liste des éléments recherchés)
11. Supprimer la tête de la liste
12. Supprimer la fin de la liste
13. Libérer votre liste
14. Afficher votre liste après chaque manipulation



Scanned with CamScanner

USMBA/ ENSAF
Structures de données

Pr. Karima AISSAOUI

# TP/TD 05 : Les piles et les files

**Objectif :** Maîtriser les manipulations des piles et files.

## Exercice 1 : les piles

Créer un système de pile

1. Créer la structure Element composée de :
    - Un entier nombre
    - Un pointeur vers l'élément suivant
2. Créer la structure Pile composée de :
    - Un élément constituant la tête de la pile
3. Créer la fonction d'empilage
4. Créer la fonction de dépilage
5. Créer la fonction d'affichage
6. Ecrire le main et appeler les différentes fonctions

## Exercice 2 : les files

Créer un système de file

7. Créer la structure Element composée de :
    - Un entier nombre
    - Un pointeur vers l'élément suivant
8. Créer la structure File composée de :
    - Un élément constituant la tête de la File
9. Créer la fonction d'enfilage
10. Créer la fonction de défilage
11. Créer la fonction d'affichage
12. Ecrire le main et appeler les différentes fonctions



Scanned with CamScanner

USMBA/ ENSAF
Structures de données

Pr. Karima AISSAOUI

# TP/TD 06 : Les arbres

**Objectif :** Simuler un arbre binaire de recherche.

## Exercice 1 : les arbres binaires de recherche

1. Définir la structure d'un arbre binaire de recherche en utilisant :
   - Un sous-arbre gauche
   - Un sous-arbre droit
   - Une racine
2. Initialiser par la suite cet arbre avec une valeur de la racine
3. Ecrire la fonction qui permet d'insérer un élément dans l'arbre (vous devez respecter les règles d'ajout déjà vues dans le cours).
4. Ecrire une fonction d'affichage de l'arbre, vous pouvez écrire une fonction pour les 3 types d'affichage vus dans le cours :
   - Pré-ordre
   - Post-ordre,
   - Symétrique
5. Ecrire la fonction qui supprime un élément de l'arbre. Etudier les 3 cas possibles des nœuds :
   - Le Si le nœud à supprimer n'a pas de fils ➔ il suffit de le supprimer.
   - Si le nœud à supprimer a un seul fils ➔ On le remplace par son fils et on supprime ce dernier.
   - Si le nœud à supprimer a deux fils ➔ On le remplace par l'extrémité du bord gauche du sous-arbre droit.
6. Ecrire une fonction qui retourne la hauteur de l'arbre.
7. Ecrire une fonction qui retourne le nombre des nœuds de l'arbre.



Scanned with CamScanner

USMBA/ ENSAF
Structures de données

Pr. Karima AISSAOUI

# TP/TD 07 : Les tables de hachage

**Objectif :** Créer une fonction et une table de hachage.

1. Définir la structure Eleve en utilisant :
   - Un tableau de caractères pour le nom
   - Un pointeur de type Eleve
2. Ecrire la fonction de hachage qui permet de retourner un entier représentant l'indice. Vous pouvez utiliser la fonction de hachage expliquée dans le cours (celle du code ASCII)
3. Créer la fonction d'initialisation de la table de hachage
4. Ecrire la fonction de remplissage. **Traiter aussi les cas de collision**
5. Ecrire une fonction d'affichage de la table de hachage



Scanned with CamScanner

USMBA/ ENSAF
Structures de données

Pr. Karima AISSAOUI

# TP/TD 08 : Les algorithmes de tri

**Objectif :** Trier les listes chaînées

1. Appliquer ces différents algorithmes de tri sur listes simplement chaînées
   - Tri par insertion
   - Tri à bulle
   - Tri par fusion
   - Tri par sélection

2. Ecrire une fonction d'affichage de la liste et appeler la avant et après chaque tri.



Scanned with CamScanner

UNIVESITE SIDI MOHAMED BEN ABDELLAH

Ecole Nationale des Sciences Appliquées

# Rappels sur les tableaux

- ***Préparé par :*** -SELLAK MOUAD
  -RHARZOZ SOUFIANE
- ***Encadrée par :*** - Pr. HRAOUI SAID


- ***Année universitaire :*** 2019/2020



Scanned with CamScanner

## ***SOMMAIRE :***


- *INTRODUCTION*
- *OBJECTIFS DU TP*
- *DEVELOPPEMEMT*
  - *Exercice 1 : Vérifier l'existence d'un entier dans un tableau.*
  - *Exercice 2 : Insérer un entier dans un tableau.*
  - *Exercice 3 : Effacer les zéros dans un tableau.*
  - *Exercice 4 : Remplissage, affichage d'une matrice ainsi que la somme de ses éléments.*
- *CONCLUSION*




Scanned with CamScanner

## *INTRODUCTION :*

***Un tableau*** *en langage C est une structure de donnée permettant de rassembler les variables du* ***même type*** *désignés par un identificateur unique, il est composé d'un ensemble de cases, chaque case est représentée par son indice, et dans chaque case on peut garder une valeur.*

*Dans ce premier TP on est devant quatre exercices à faire, en ce qui concerne des opérations sur des tableaux telles que l'enquête, l'insertion, l'élimination d'un entier, ainsi que la familiarisation avec les matrices ou les tableaux bidimensionnels (déclaration, remplissage, affichage).*

## *OBJECTIFS DU TP :*

*Le but de ce TP consiste à rappeler les notions fondamentales des tableaux en langage C, que ce soit des tableaux unidimensionnels ou bidimensionnels, afin d'avoir une aisance à les utiliser en ce qui suit, que ce soit les structures, les listes chainées …etc.*

## *DEVELOPPEMEMT :*

*Dans cet axe on va essayer de copier l'énoncée, le programme, et l'exécution de chaque exercice.*



Scanned with CamScanner

## Exercice 1 : Vérifier l'existence d'un entier dans un tableau.

### Enoncé :

*Soit le tableau suivant :*

*int [] tab = {12, 15, 13, 10, 8, 9, 13, 14} ;*

***1)**-Ecrire un programme qui saisit un entier au clavier et qui recherche si cet entier appartient au tableau (réponse de type oui/non).*

***2)**-Ecrire un programme qui saisit un entier au clavier et qui recherche si cet entier appartient au tableau donné au début de l'exercice. Au cas où la réponse est positive, l'indice de cet entier dans le tableau est affiché. S'il y a plusieurs occurrences, le dernier indice est affiché.*

### Réponse :

#### Question1 :

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

main(){
     int tab[]={12,15,13,10,8,9,13,14};
     int i,a,c=0;
     printf("Entrer un entier ! ");
     scanf("%d",&a);
     for(i=0;i<7;i++){
          if(tab[i]==a) c=1;
     }
     if(c==1)
     printf("Oui");
     else
     printf("Non ");
return 0;
}
```

### Compilation :



Scanned with CamScanner

```
Entrer un entier ! 10
Oui
```

```
Entrer un entier ! 11
Non
```

## Question2 :

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
main(){
int tab[]={12,15,13,10,8,9,13,14};
    int i,a,c=0,j=1;
    printf("Entrer un entier ! ");
    scanf("%d",&a);
    for(i=0;i<7;i++){
        if(tab[i]==a)   {
        c=1;
        j=i;
            }
        }
    if(c==1)
    {
        printf("La valeur %d existe  et son indice est %d.",a,j+1);
    }
    else
    printf("Non! ",a);
return 0;
}
```

**Compilation :**

```
Entrer un entier ! 9
La valeur 9 existe  et son indice est 6.
```

## Exercice 2 : Insérer un entier dans un groupe :

### Enoncé :

*Ecrire un programme qui saisit un entier et un indice et met cet entier dans le tableau à cet indice. Il faudra vérifier que l'indice correspond bien à une case du tableau (par exemple 0 ou 3 sont corrects, mais -1 ou 100 ne sont pas des indices corrects pour ce tableau). Le programme affichera le contenu du tableau avant et après cette transformation.*



Scanned with CamScanner

## ***Réponse :***

```c
#include<stdio.h>
main(){
int T[100];
int n,a,j,i,t;
printf("Entrer la dimension de votre tableau: ");
scanf("%d",&n);
for(i=0;i<n;i++){
    printf("T[%d]=",i+1);
    scanf("%d",&T[i]); }
printf("\n \t Avant la transformation : ");
for(i=0;i<n;i++){
printf("%6d", T[i]);    }
printf("\n \n Entrer l'element que vous voulez inserer dans votre tableau: ");
scanf("%d",&a);
do {
printf("\n Entrer son indice : ");
scanf("%d",&j);
if (j<0 || j>n)
printf("\n rentrer son indice ! ");
 }while(j<0 || j>n);
 for(i=0;i<n;i++){
    if(j==i) T[j]=a;}
printf("\n \t Apres la transformation : ");
for(i=0;i<n;i++)
printf("%6d", T[i]);
}
```

## ***Compilation :***

```
Entrer la dimension de votre tableau: 4
T[1]=1
T[2]=7
T[3]=4
T[4]=3

        Avant la transformation :      1     7     4     3

 Entrer l'element que vous voulez inserer dans votre tableau: 2233

 Entrer son indice : 1

        Apres la transformation :      1  2233     4     3
--------------------------------
```



Scanned with CamScanner

## Exercice 3 : Effacer les zéros dans un tableau.:

### Enoncé :

*Ecrire un programme qui saisit la dimension N d'un tableau de int (le tableau est initialement défini avec une taille maximum MAX que N ne doit pas excéder) remplit le tableau par des valeurs entrées au clavier et l'affiche. Le programme doit ensuite effacer toutes les occurrences de la valeur 0 dans le tableau, insérer les éléments restants et afficher le tableau ainsi modifié.*

### Réponse :

```c
#include<stdio.h>
#define  Max 100
main(){
int T[Max],TM[Max];
int N,i,M,j=0;
do{
    printf("Entrer la dimension de votre tableau ! ");
    scanf("%d",&N);
    if(N>Max || N<0)
    printf("Vous devez entrer une dimension inférieur a %d",Max);
}while(N>Max || N<0);
for(i=0;i<N;i++){
    printf("T[%d]=",i);
    scanf("%d",&T[i]);}
printf("\n \n \t Affichage : ");
for(i=0;i<N;i++){
printf("%6d", T[i]);    }
for(i=0;i<N;i++){
    if (T[i]!=0)
     {TM[j]=T[i];
     j++; } }
printf("\n \n  \t Apres la modification : ");
for(i=0;i<j;i++)
printf("%d \t", TM[i]);
return 0;
}
```



Scanned with CamScanner

## Compilation :

```
Entrer la dimension de votre tableau ! 4
T[0]=1
T[1]=2
T[2]=0
T[3]=4


         Affichage :      1     2     0     4

         Apres la modification : 1      2       4
```

## Exercice 4 : Remplissage, affichage d'une matrice ainsi que la somme de ses éléments.

### Enoncé :

*Ecrire un programme qui lit les dimensions L et C d'un tableau T à deux dimensions du type int (dimensions maximales : 50 lignes et 50 colonnes). Remplir le tableau par des valeurs entrées au clavier et afficher le tableau ainsi que la somme de tous ses éléments. :*

### Réponse :

```c
#include<stdio.h>
main(){
int T[50][50];
int i,j,L,C,S=0;
printf("Entrer la dimension de votre matrice: \n  ");
printf("Nombres des lignes : ");
scanf("%d",&L);
printf("\n Nombres des colonnes : ");
scanf("%d",&C);
printf("\n \n Le remplissage: \n \n ");
for(i=0;i<L;i++){
     for(j=0;j<C;j++){
          printf("T[%d][%d]=",i+1,j+1);
          scanf("%d",&T[i][j]); } }
printf("\n \n L'affichge : \n \n ");
for(i=0;i<L;i++){
     for(j=0;j<C;j++)
          printf("%4d",T[i][j]);
          printf("\n \n "); }
for(i=0;i<L;i++){
     for(j=0;j<C;j++){
     S+=T[i][j]; } }
printf("\n La somme des elements est %d \n \n  ",S);
return 0;
}
```



Scanned with CamScanner

## *Compilation :*

```
Entrer la dimension de votre matrice:
  Nombres des lignes : 3

 Nombres des colonnes : 3


 Le remplissage:

 T[1][1]=1
T[1][2]=1
T[1][3]=1
T[2][1]=1
T[2][2]=1
T[2][3]=1
T[3][1]=1
T[3][2]=1
T[3][3]=1


 L'affichge :

    1   1   1

    1   1   1

    1   1   1


 La somme des elements est 9
```



Scanned with CamScanner

## ***CONCLUSION :***

*Ce TP nous a donné comme opportunité l'occasion d'utiliser les tableaux et similaire leurs performance et utilité dans des programmes en langage performant C, d'autre part nous avons appris comment travailler avec des tableaux bidimensionnels (matrices), et effectuer des instructions sur eux selon la résolution nécessite.*

*Dans ce qui suit on va étudier les pointeurs et leurs importantes utilités dans la résolution des programme longs.*

Scanned with CamScanner

UNIVESITE SIDI MOHAMED BEN ABDELLAH

Ecole Nationale des Sciences Appliquées

ENSA Fès

# STRUCURES DES DONNEES EN C

## TP2 : Pointeurs et allocation dynamique


- **Préparé par :** -SELLAK MOUAD - N°207
  -RHARZOZ SOUFIANE - N°198
- **Encadré par :** - Pr. HRAOUI SAID


- **Année universitaire :** 2019/2020



Scanned with CamScanner

## ***INTRODUCTION :***

*Après avoir étudier et réviser la notion des tableaux on a passé à découvrir une autre notion très importante dans le langage C, il s'agit des pointeurs, que désigne donc un pointeur ? et quelle est son utilité ?*

*Un pointeur est une variable qui contient l'adresse d'une autre variable sur laquelle il pointe, Il a lui-même une adresse propre, car chaque variable dans la mémoire est connue par une référence spécifiée.*

*La déclaration d'un pointeur :* ***type *identificateur ;***

*L'opérateur* * *s'appelle l'opérateur de déréférencement, c'est-à-dire il permet d'identifier le contenu de la variable sur laquelle notre pointeur pointe.*

*Vu leurs importances, les pointeurs ont plusieurs utilisations, et parmi leurs applications il y a ce qu'on appelle* ***l'allocation dynamique****, c'est-à-dire le fait d'allouer un espace mémoire à utiliser plus tard d'une manière dynamique, et ça bien évidement permet d'éviter le gaspillage de la mémoire qui se pose en cas de traitement statique.*

## ***OBJECTIFS DU TP :***

*Le but de ce TP consiste à maitriser bien la notion des pointeurs et saisir leurs performances et utilités.*



Scanned with CamScanner

## DEVELOPPEMEMT :

*L'opérateur « **sizeof** » permet de consulter la taille de chaque type prédéfini en langage C, il est important surtout dans l'allocation dynamique.*

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

 // Consulter la taille (en -byte-) des types prédifinies en C :
main(){
printf("int:%d \n float:%d \n char:%d ",sizeof(int),sizeof(float),sizeof(char));
printf(" \n double:%d \n void:%d",sizeof(double),sizeof(void));
return 0;
}
```

```
int:4
 float:4
 char:1
 double:8
 void:1
```

### Exercice 1 :

*- Déclarer un pointeur sur int et l'initialiser par le pointeur NULL.*

*-Déclarer un pointeur sur pointeur sur int.*

*- Déclarer une variable de type int.*

*- Initialiser le pointeur sur int par l'adresse de cette variable.*

*- Initialiser le pointeur sur pointeur sur int par l'adresse du pointeur sur int.*

*- Allouer dynamiquement un espace de mémoire suffisant pour contenir une variable de type int, et stocker son adresse dans le pointeur sur int.*

Scanned with CamScanner

***Réponse :***

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
main(){
// Déclaration d'un pointeur et l'initialiser a la valeur NULL:
int *ptr=NULL;  // printf("%p",ptr); => 000000000 (%p pour afficher l'adresse.)
// Déclaration d'un pointeur sur un pointeur :
int **p;
// Déclaration d'une variable:
int a;
// initialisation du pointeur p par l'adresse de a:
ptr=&a;
// initialisation du pointeur t par l'adresse du poiteur p :
p=&ptr;
// alouer dynamiquement un espace de memoire:
ptr=(int *)malloc(sizeof(a));
ptr=&a;
return 0;
}
```

***Exercice 2 :***

*- Déclarer deux pointeurs sur float .*

*- Déclarer deux variables de type float .*

*-Initialiser chacun des pointeurs avec les adresses des deux variables float .*

*-Affecter la valeur 12.5 à la première variable en utilisant son pointeur.*

*- Affecter la valeur 5.76 à la deuxième variable en utilisant son pointeur.*

*- Échanger le contenu des deux variables en utilisant leurs pointeurs.*



Scanned with CamScanner

## ***Réponse :***

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
main(){
    // Déclaration
float *p1,*p2;
float r1,r2,t;
// Initialisation
p1=&r1;
p2=&r2;
// Affectation
*p1=12.5; // <==> *p1=r1=12.5
*p2=5.76; // <==> *p2=r2=5.76
printf("Avant l'echange ! r1= %.2f \t r2=%.2f", *p1,*p2);
// Permutation
t=*p1;
*p1=*p2;
*p2=t;
// Affichage
printf(" \n \n Apres l'echange ! r1=%.2f \t r2=%.2f", *p1,*p2);
return 0;
}
```

***Compilation :***

```
Avant l'echange ! r1= 12.50      r2=5.76

 Apres l'echange ! r1=5.76       r2=12.50
```

***Exercice 3 :***

*Écrire une procédure "permute" qui permet de permuter les valeurs de deux variables entières et écrire un programme dans lequel on saisira deux nombres entiers avant de faire appel à cette procédure et d'afficher le contenu de ces variables afin de vérifier la permutation.*



Scanned with CamScanner

## ***Réponse :***

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
// prototype:
void permute(int *p_a, int *p_b);

main(){
int x,y;
printf("Entrer la valeur de x : ");
scanf("%d",&x);
printf("Entrer la valeur de y : ");
scanf("%d",&y);
printf("Avant l'echange : x=%d \t y=%d", x,y);
// Appel de la procédure :
permute(&x,&y); // Permutation par adresse et non pas par parametre !
printf(" \n \n Apres l'echange : x=%d \t y=%d", x,y);
return 0;
}
// Définition de la procédure:
void permute(int *p_a, int *p_b){
    int t;
    t=*p_a;
    *p_a=*p_b;
    *p_b=t;
}
```

### ***Compilation :***

```
Entrer la valeur de x : 10
Entrer la valeur de y : 20
Avant l'echange : x=10   y=20

 Apres l'echange : x=20          y=10
```

### ***Exercice 4 :***

*Utiliser l'allocation dynamique pour créer un tableau de 15 variables de type float. Initialiser ce tableau avec des zéros, puis libérer la mémoire allouée si l'allocation a été effectuée avec succès.*



Scanned with CamScanner

## ***Réponse :***

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
main(){
    // Déclaration
float *tab;
// Allocation de la mémoire pour stoker 15 élements d'un taleau:
tab=(float *)malloc(15*sizeof(float));
// Allocation de la mémoire et l'initialisation des élements du tableau par des 0:
tab=(float *)calloc(15,sizeof(float));
// Libérer la mémoire éffectué en succes:
if(tab==NULL)
printf("La memoire est pas alouee ! ");
else
free(tab); // on l'utilise une seule fois !
return 0;
}
```

## ***Exercice 5 :***

*-Déclarer un double pointeur sur float, puis utiliser une fonction pour effectuer l'allocation dynamique d'un tableau de 15 variables de type float.*

*- Utiliser une fonction pour initialiser ce tableau avec des zéros si l'allocation a été effectuée avec succès.*

*-Puis libérer la mémoire allouée toujours si l'allocation a été effectuée avec succès.*

## ***Réponse :***

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
main(){
    // Déclaration
float **p;
// Allocation de la mémoire pour stoker 15 élements d'un taleau
*p=(float *)malloc(15*sizeof(float));
// Allocation de la mémoire et l'initialisation des élements du tableau par des 0:
*p=(float *)calloc(15,sizeof(float));
// Libérer la mémoire éffectué en succes
free(*p);
return 0;
}
```

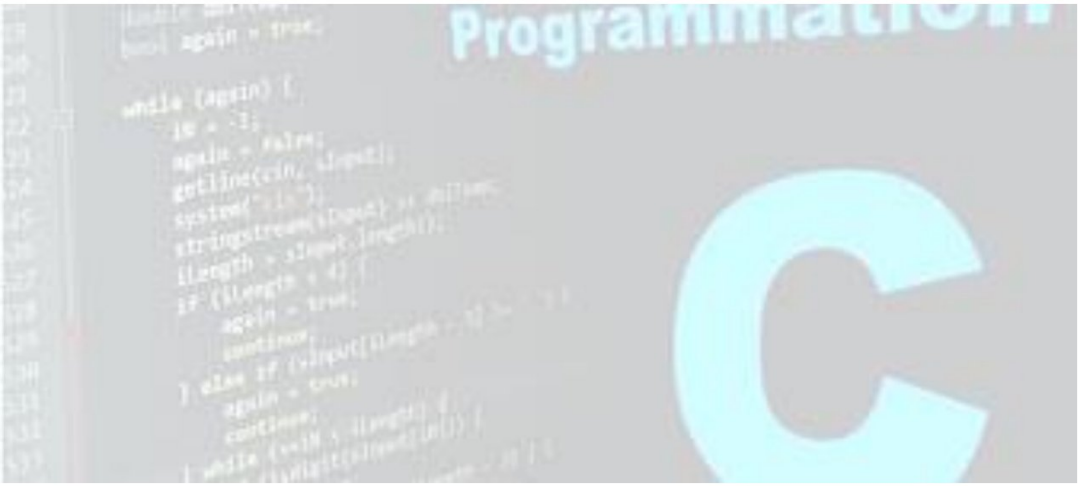


Scanned with CamScanner

## ***CONCLUSION :***

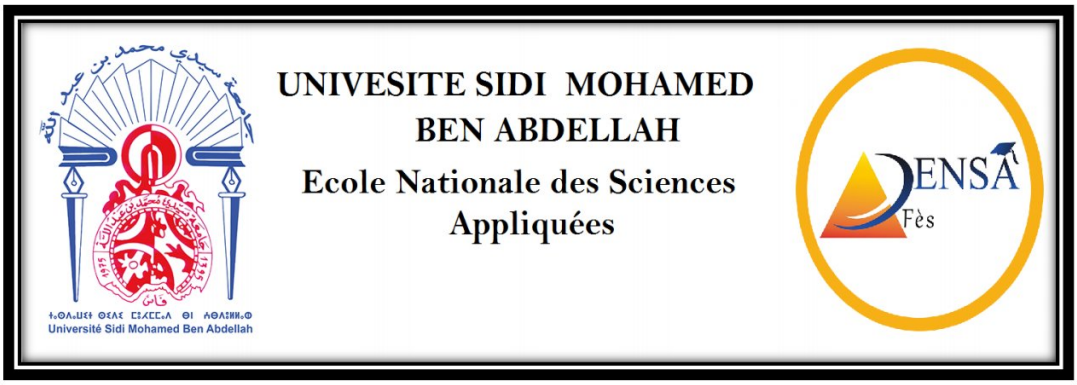
*Ce TP nous a donné comme opportunité l'occasion de comprendre les pointeurs et saisir leurs performance et utilité dans des programmes en langage performant C, surtout dans l'allocation dynamique, d'autre part nous avons appris comment allouer dynamiquement un espace mémoire et le contrôler en fonction du besoin.*

*Dans ce qui suit on va plus s'approfondir dans l'utilisation des pointeurs et découvrir encore leurs utilités dans la résolution de plusieurs problèmes.*






Scanned with CamScanner




# STRUCURES DES DONNEES EN C

## TP3 : Listes simplement chainées

***Préparé par :*** *-SELLAK MOUAD - N°207*

*-RHARZOZ SOUFIANE - N°198*

*Groupe 4-2*

***Encadré par :*** *- Pr. HRAOUI SAID*

***Année universitaire :*** *2019/2020*

Scanned with CamScanner

## →Introduction :

*Dans la dernière séance du TP, il apparait que les étudiants ont pu maitriser les listes chaînées, maintenant ils sont prêts pour entamer les listes chainées.*

*Une liste chainée représente une façon d'organiser les données et les éléments reliés entre elles par des pointeurs d'une manière plus flexible. On peut ajouter et enlever des éléments dans différentes endroits et à chaque instant sans devoir récréer la liste entière.*

## →But :

*Le but de ce TP est :*

- *Savoir une structure d'éléments et de liste.*
- *Initialiser la liste avec une valeur.*
- *Ajouter des éléments de liste au début et à la fin.*
- *Supprimer des éléments de liste au début et à la fin.*
- *Afficher la liste après manipulation*

## →Exercices :

### Q1 : créer la structure ELEMENT

*On va créer cette structure contenant une valeur de type* **int** *et un pointeur vers un pointeur de même type.*

Scanned with CamScanner

*C'est celui-ci qui va permettre de relier ces éléments entre eux.*

```
struct Element{
    int valeur;
    Element *p_suivant;
};
```

## Q2 : créer la structure LISTE :

*Cette structure comporte une longueur de type* **SHORT INT** *et un élément tête de liste. Elle permet de contrôler les listes chainées.*

```
struct List{
    short int longueur;
    Element *p_tete;
};
```

## Q3 : Initialisation de la liste avec un élément de valeur « int valeur » :

*La première fonction à réaliser est celle de l'initialisation dont il faut créer le premier élément de liste.*

Scanned with CamScanner

```
List* Initialiser( int valeur){
    Element *element=(Element*)malloc(sizeof(Element));
    List *liste=(List*)malloc(sizeof(List));
if(liste==NULL || element==NULL)
    exit(EXIT_FAILURE);

    element->valeur=valeur;
    element->p_suivant=NULL;

    liste->longueur=1;
    liste->p_tete=element;
}
```

## Q4 : Ajouter un élément en début de la liste :

*La deuxième fonction est l'ajout au début.*

*Lors de cette fonction : Nous allons créer un élément, lui assigner la valeur que l'on veut ajouter, raccorder cet élément à la liste passée en paramètre. Et finalement, on doit assigner à* ***p_suivant*** *l'adresse du 1$^{er}$élément de la liste passée en paramètre.*

Scanned with CamScanner

```
void AjouTete(List *liste, int valeur){
    Element *element=(Element *)malloc(sizeof(Element));

    if(element==NULL || liste==NULL )
    exit(EXIT_FAILURE);

    element->valeur=valeur;

    element->p_suivant=liste->p_tete;
    liste->p_tete=element;

    liste->longueur++;
}
```

## Q5: Ajouter un élément en fin de liste :

*La troisième fonction est l'ajout en fin.*

*Lors de cette fonction : il faut d'abord créer un nouvel élément, Lui assigner sa valeur, mettre l'adresse de l'élément suivant NULL. Et finalement, il faut pointer le dernier élément de liste originale sur le nouveau élément que nous venons de créer.*

```
void AjouFin(List* liste, int valeur){
    Element *element=(Element*)malloc(sizeof(element));
    Element *tmp=liste->p_tete;
    if(element==NULL || liste==NULL)
    exit(EXIT_FAILURE);

    element->valeur=valeur;
    element->p_suivant=NULL;

    while(tmp->p_suivant!=NULL)
    tmp=tmp->p_suivant;

    tmp->p_suivant=element;
    liste->longueur++;
}
```

Scanned with CamScanner

## Q6: Afficher votre liste après chaque manipulation :

*Pour bien visualiser ce que comporte cette liste. On va créer une fonction d'affichage de la manipulation précédente*

```
void Afficher(List *liste){

    Element *parc=liste->p_tete;
    while(parc!=NULL){
        printf(" %d -> ",parc->valeur);
        parc=parc->p_suivant;
     }
     printf(" NULL \n \n  ");
}
```

## Q7 : Supprimer le premier élément de liste :

*Tout d'abord on va mettre l'élément à supprimer dans une variable temporaire, puis le premier élément de la liste devient le suivant d'élément à supprimer, et on fait liberer l'élément à supprimer en décrementant la longueur de la liste.*

Scanned with CamScanner

```
void SuppTete(List *liste){
    if(liste==NULL)
    exit(EXIT_FAILURE);

    Element *asupprimer=liste->p_tete;
    liste->p_tete=asupprimer->p_suivant;
    liste->longueur-=1;
    free(asupprimer);
}
```

## Q8 : Supprimer le dernier élément de liste :

*De la même philosophie on crée un élément temporaire qui va parcourir la liste jusqu'arriver au dernier élément puis libérer ce dernier après le placer dans un élément temporaire.*

```
void SuppFin(List *liste){
    if(liste==NULL)
    exit(EXIT_FAILURE);

    Element *asupprimer=liste->p_tete;
    Element *tmp;
    while(asupprimer->p_suivant!=NULL){
        tmp=asupprimer;
        asupprimer=asupprimer->p_suivant;
    }
    tmp->p_suivant=NULL;
    liste->longueur-=1;
    free(asupprimer);
}
```

## Fonctions supplémentaires :

Scanned with CamScanner

```c
// Ajouter un élément au milieu selon la position:
void AjoutMili(List *liste, int valeur, int position){
    Element *New=(Element*)malloc(sizeof(Element));
    int i=1;
    if(New==NULL || liste==NULL){
        exit(EXIT_FAILURE);
    }
    New->valeur=valeur;
    Element *tmp=liste->p_tete;
    while(tmp!=NULL && i<position-1){
        tmp=tmp->p_suivant;
        i++;
    }
New->p_suivant=tmp->p_suivant;
tmp->p_suivant=New;
}
```

```c
// Rechercher une valeur et la modifier:
void RechMod(List *liste, int amodifier, int Nvaleur){
    Element *tmp=liste->p_tete;
    while(tmp!=NULL){
        if(tmp->valeur==amodifier)
        tmp->valeur=Nvaleur;
        tmp=tmp->p_suivant;

    }
}
```

Scanned with CamScanner

```
// Supprimer de la liste:
void  Delete(List *liste){
    if(liste!=NULL){
        Element *tmp=liste->p_tete;
        Element *fre;
        while(tmp!=NULL){
            fre=tmp;
            tmp=tmp->p_suivant;
            free(fre);

        }
    }
    liste=NULL;
}
```

# La fonction main :

```
main(){
List *liste;

liste=Initialiser(1);
Afficher(liste);

AjouTete(liste,7);
 Afficher(liste);

AjouFin(liste,13);
Afficher(liste);

SuppTete(liste);
Afficher(liste);

SuppFin(liste);
Afficher(liste);

AjoutMili(liste,1111,2);
Afficher(liste);

RechMod(liste, 1, 12888);
Afficher(liste);

Delete(liste);
```

Scanned with CamScanner

***Compilation :***

```
1 ->  NULL
  7 ->  1 ->  NULL
  7 ->  1 ->  13 ->  NULL
  1 ->  13 ->  NULL
  1 ->  NULL
  1 ->  1111 ->  NULL
  12888 ->  1111 ->  NULL
```

## CONCLUSION :

*Pour finir, nous avons atteint l'objectif principal du TP en maitrisant les listes chainées simples et différentes notions de ces listes tel que la création, l'initialisation, l'ajout et l'affichage finale.*

Scanned with CamScanner

UNIVESITE SIDI MOHAMED
BEN ABDELLAH
Ecole Nationale des Sciences
Appliquées

# STRUCURES DES DONNEES EN C

## TP4 : Listes doublement chainées

➢ **Préparé par :**-SELLAK MOUAD - N°207
-RHARZOZ SOUFIANE - N°198

➢ **Encadré par** : - Pr. HRAOUI SAID

➢ **Année universitaire : 2019/2020**



Scanned with CamScanner

## INTRODUCTION :

*Après avoir étudier la notion des listes simplement chainées on a passé à découvrir une autre notion très importante dans le langage C, il s'agit des listes doublement chainées, que désigne donc une liste doublement chainée ? et quelle est son utilité ?*

*Une liste doublement chainée est une liste qui contient une donnée et deux pointeurs, l'un sur l'élément suivant, et l'autre sur l'élément précédent.*

*Dans ce TP on va découvrir vraiment l'utilité de ces listes.*

## OBJECTIFS DU TP :

*Le but de ce TP consiste à maitriser bien la notion des listes doublement chainées et saisir leurs performances et avantages.*

## DEVELOPPEMEMT :

### Q2 : créer la structure node:

*On va créer cette structure contenant une valeur de type int et un pointeur vers un élément suivant et un autre pointeur vers un élément précèdent.*

```
struct node{
    int data;
    node *p_next;
    node *p_prev;
};
```



Scanned with CamScanner

## *Q2 : créer la structure dlist :*

*Cette structure comporte une longueur de type SHORT INT et un élément tête de liste et un autre élément représentant la queue de la liste. Elles permettent de contrôler les listes chainées.*

```
struct dlist{
    short int longueur;
    node *p_head;
    node *p_teal;
};
```

## *Q3 : Allouer et initialiser une nouvelle liste :*

*On veut créer une nouvelle liste.* ***malloc*** *sert à réserver de l'espace mémoire pour cette liste.*

```
dlist *New(){
    dlist *list=(dlist*)malloc(sizeof(dlist));
    if(list==NULL){
        exit(EXIT_FAILURE);
    }
    list->longueur=0;
    list->p_head=NULL;
    list->p_teal=NULL;
    return list;
}
```



Scanned with CamScanner

```
dlist *Initialisation(dlist *liste, int data){
    node *New=(node*)malloc(sizeof(node));

    New->data=data;
    New->p_next=NULL;
    New->p_prev=NULL;

    liste->longueur=1;
    liste->p_head=New;
    liste->p_teal=New;

    return liste;
}
```

## *Q4: Ajouter un node en fin de la liste :*

*Tout d'abord, il faut vérifier que la liste n'est pas NULL. Si elle ne l'est pas, nous allons créer un nouvel élément (nouveau node). Enfin, nous incrémentons notre champ Longueur de notre liste puis nous retournons la liste. Tout ceci constitue alors l'algorithme d'ajout en fin de liste.*



Scanned with CamScanner

```c
void AjoutFin(dlist *list, int Ndata){
    node *New=(node*)malloc(sizeof(node));
    if(list==NULL || New==NULL ){
        exit(EXIT_FAILURE);
    }
    New->data=Ndata;
    New->p_next=NULL;

    if(list->p_teal==NULL){
    New->p_prev=NULL;
    list->p_head=New;
    list->p_teal=New;
    }
    else{
    New->p_prev=list->p_teal;
    list->p_teal->p_next=New;
    list->p_teal=New;
    }
    list->longueur++;
}
```

## Q5 : Ajouter un node en début de la liste :

*Pour ajouter un élément en début de liste, il faut utiliser exactement le même procédé que pour l'ajout en fin de la liste.*

```c
void AjoutDebut(dlist *list, int data){
    node *New=(node*)malloc(sizeof(node));
    if(list==NULL || New==NULL){
        exit(EXIT_FAILURE);
    }
    New->data=data;
    New->p_prev=NULL;
    New->p_next=list->p_head;
    list->p_head=New;
    list->longueur++;
}
```



Scanned with CamScanner

## Q6: Ajouter un node selon sa position :

```
void AjoutPos(dlist* liste, int data, int position){
    node *New=(node*)malloc(sizeof(node));
    int i=1;
    if(liste==NULL || New==NULL){
    exit(1); }
    New->data=data;
    node *tmp=liste->p_head;
    while(tmp!=NULL && i<=position){
        if(position==i){
            if(tmp->p_prev==NULL){
            AjoutDebut(liste, data);
            }
            else if(tmp->p_next==NULL){
            AjoutFin(liste,data); }
            else{
            New->p_next=tmp->p_next;
            New->p_prev=tmp;
            tmp->p_next->p_prev=New;
            tmp->p_next=New;
            liste->longueur++;
            }   }
        else
        tmp=tmp->p_next;
        i++;    }
}
```

## Q7 : Supprimer un node selon sa valeur :



Scanned with CamScanner

```
void SuppVal(dlist *liste, int valeur){
    node* tmp=liste->p_head;
    int found=0;
    while(tmp!=NULL && !found){
        if(tmp->data==valeur){
            {
                if(tmp->p_prev==NULL){
                    liste->p_head=tmp->p_next;
                    liste->p_head->p_prev=NULL;
                 }
                else if(tmp->p_next==NULL){
                    liste->p_teal=tmp->p_prev;
                    liste->p_teal->p_next=NULL;
                }
                else{
                    tmp->p_next->p_prev=tmp->p_prev;
                    tmp->p_prev->p_next=tmp->p_next;
                }
                free(tmp);
                liste->longueur--;
                found=1; }
        }
        else
        tmp=tmp->p_next;  }
}
```

## *Q8 : Supprimer un node selon sa position:*



Scanned with CamScanner

```
void SuppPos(dlist *liste, int position){
    int i=1;

    node *tmp=liste->p_head;
    while(tmp!=NULL && i<=position){
        if(position==i){
            if(tmp->p_prev==NULL)
            SuppTete(liste);
            else if(tmp->p_next==NULL)
            SuppFin(liste);
            else{
                tmp->p_prev->p_next=tmp->p_next;
                tmp->p_next->p_prev=tmp->p_prev;
                liste->longueur--;
                free(tmp);
            }
        }
        else
        tmp=tmp->p_next;
        i++;
    }
}
```

## Q9 : recherche d'un node selon la valeur :

```
dlist* RechElement(dlist* liste, int valeur){
    dlist* list=New();
    list=Initialisation(list,0);
    node* tmp=liste->p_head;
    int found=1;
    while(tmp!=NULL && !found ){
        if(tmp->data==valeur){
            AjoutDebut(list,valeur);
            found=0;
        }
        else
        tmp=tmp->p_next;
    }
    return list;
}
```



Scanned with CamScanner

## Q10 : recherche d'un ensemble de node selon la valeur:

*Il faut créer une liste. Si on trouve un élément, on l'initialise. Puis ajouter les éléments trouvés dedans.*

```
dlist* RechElements(dlist *liste, int valeur){
    dlist* list=NULL;
    node *tmp=liste->p_head;
    while(tmp!=NULL){
        if(tmp->data==valeur){
            if(list==NULL)
        list=New();
        AjoutDebut(list,valeur);
            }
        tmp=tmp->p_next;
    }
    return list;
}
```

## Q11 : Supprimer le premier élément de liste :

```
void SuppTete(dlist* liste){
    node *tmp=liste->p_head;
    node*fre;
    fre=tmp;
    tmp=tmp->p_next;
    tmp->p_prev=NULL;
    liste->p_head=tmp;
    free(fre);
}
```

## Q12 : Supprimer le dernier élément de liste :



Scanned with CamScanner

```c
void SuppFin(dlist* liste){
    node *tmp=liste->p_teal;
    node *fre=tmp;
    liste->p_teal=tmp->p_prev;
    tmp->p_prev->p_next=NULL;
    free(fre);
}
```

## Q13 : libérer votre liste :

```c
void Liberer(dlist* liste){
    node *tmp=liste->p_head;
    node *fre;
    while(tmp!=NULL){
        fre=tmp;
        tmp=tmp->p_next;
        free(fre);
    }
    liste->p_head=NULL;
    liste->p_teal=NULL;
}
```

## Q14: Afficher votre liste après chaque manipulation :

```c
void Afficher(dlist *liste){
    node* tmp=liste->p_head;
    while(tmp!=NULL){
        printf("%d <=> ",tmp->data );
        tmp=tmp->p_next;
    }
    printf(" NULL \n \n");
}
```



Scanned with CamScanner

```c
main(){
dlist *liste;
dlist *list1,*list2;
liste=New();
printf(" Initialisation: \t ");
liste=Initialisation(liste, 1);
Afficher(liste);
printf("\n \n Ajouter au debut: \t ");
AjoutDebut(liste, 12);
AjoutDebut(liste, 12);
AjoutDebut(liste, 12);
AjoutDebut(liste, 144);
Afficher(liste);
printf("\n \n Ajouter en fin: \t ");
AjoutFin(liste, 666);
AjoutFin(liste, 22);
AjoutFin(liste, 62);
Afficher(liste);
printf("\n \n Ajouter en position: \t ");
AjoutPos(liste,23,1);
Afficher(liste);
printf("\n \n supprimer par valeur: \t ");
SuppVal(liste,22);
Afficher(liste);
printf("\n \n supprimer la tete: \t ");
SuppTete(liste);
Afficher(liste);
printf("\n  \n supprimer la queue: \t ");
SuppFin(liste);
Afficher(liste);
printf("\n \n L'element recherche: \t");
list1=RechElement(liste,666);
Afficher(list1);
printf("\n \n Liste des elements recherches: \t");
list2=RechElements(liste,12);
Afficher(list2);
printf("\n \n La liste apres la liberation: \t ");
Liberer(liste);
Afficher(liste);

}
```



Scanned with CamScanner

## Compilation :

```
Initialisation:         1 <=>  NULL

Ajouter au debut:       144 <=> 12 <=> 12 <=> 12 <=> 1 <=>  NULL

Ajouter en fin:         144 <=> 12 <=> 12 <=> 12 <=> 1 <=> 666 <=> 22 <=> 62 <=>  NULL

Ajouter en position:    23 <=> 144 <=> 12 <=> 12 <=> 12 <=> 1 <=> 666 <=> 22 <=> 62 <=>  NULL

supprimer par valeur:   23 <=> 144 <=> 12 <=> 12 <=> 12 <=> 1 <=> 666 <=> 62 <=>  NULL

supprimer la tete:      144 <=> 12 <=> 12 <=> 12 <=> 1 <=> 666 <=> 62 <=>  NULL

supprimer la queue:     144 <=> 12 <=> 12 <=> 12 <=> 1 <=> 666 <=>  NULL

L'element recherche:   666 <=>  NULL

Liste des elements recherches:          12 <=> 12 <=> 12 <=>  NULL

La liste apres la liberation:    NULL
```

## CONCLUSION :

*Ce TP nous a donné l'occasion de comprendre les listes doublement chainées et saisir leurs performances et utilités dans des programmes en langage performant C, surtout en ce qui concerne l'initialisation, l'insertion, la suppression, la recherche des éléments.*

*Dans ce qui suit on va plus s'approfondir dans d'autres structures des données telles que les piles, les files, les arbres … et découvrir encore leurs utilités dans la résolution de plusieurs problèmes.*



Scanned with CamScanner

UNIVESITE SIDI MOHAMED BEN ABDELLAH

Ecole Nationale des Sciences Appliquées

# STRUCURES DES DONNEES EN C

## TP5 :Piles et Files


- **Préparé par :**-SELLAK MOUAD - N°207

  -RHARZOZ SOUFIANE - N°198

- **Encadré par :** - Pr. HRAOUI SAID


- **Année universitaire : 2019/2020**



Scanned with CamScanner

## INTRODUCTION :

*Les piles et les files sont très utiles pour des programmes qui doivent traiter des données qui arrivent au fur et à mesure.*

## OBJECTIFS DU TP :

*Le but de ce TP consiste à maitriser bien la notion des listes doublement chainées et saisir leurs performances et utilités.*

## DEVELOPPEMEMT :

# Les piles :

### 1)-Créer la structure Element :

*Chaque élément de la pile aura une structure identique à celle d'une liste chaînée :*

```
struct Element {
    int valeur;
    Element *suivant;
};
```

### 2)-Créer la structure Pile :

*La structure de contrôle contiendra l'adresse du premier élément de la pile.*



Scanned with CamScanner

```c
struct Pile {
    Element *premier;
};
```

***La fonction d'initialisation :***

```c
Pile *Initialiser(int valeur){
    Element *nouveau=(Element*)malloc(sizeof(Element));
    Pile *pile=(Pile*)malloc(sizeof(Pile));
    if(nouveau==NULL || pile==NULL){
        exit(EXIT_FAILURE);
    }
    nouveau->valeur=valeur;
    nouveau->suivant=NULL;

    pile->premier=nouveau;
    return pile;
}
```

## *3)-La fonction d'empilage :*

*La fonction « empiler » doit prendre en paramètre la structure de contrôle de la pile (de type Pile) ainsi que le nouveau nombre à stocker.*

```c
void Empilage(Pile *pile, int valeur){
    Element *nouveau=(Element*)malloc(sizeof(Element));
    if(pile == NULL || nouveau == NULL){
        exit(EXIT_FAILURE);
    }
    nouveau->valeur=valeur;
    nouveau->suivant=pile->premier;
    pile->premier=nouveau;
}
```



Scanned with CamScanner

## 4)-La fonction de dépilage :

*Notre fonction « depiler » va donc retourner un int correspondant au nombre qui se trouvait en tête de pile.*

```
int Depilage(Pile *pile){
 Element *nouveau=pile->premier;
 if(nouveau==NULL){
    exit (EXIT_FAILURE);
 }
int nombreDepile=0;
 Element *elementDepile=pile->premier;
 if(pile!=NULL && pile->premier!=NULL){
    nombreDepile=elementDepile->valeur;
    pile->premier=elementDepile->suivant;
    free(elementDepile);
 }
 return nombreDepile;
}
```

## 5)-La fonction d'affichage :

```
void Affichage(Pile *pile) {
    Element *actuel=pile->premier;
    if(pile==NULL || actuel ==NULL){
        exit(EXIT_FAILURE);
    }
    while(actuel!=NULL){
        printf("\n \n  %d ",actuel->valeur);
        actuel=actuel->suivant;
    }
    printf(" \n \n  NULL \n \n");
}
```



Scanned with CamScanner

## 6)-La fonction principale :

```c
main(){

Pile *pile;
pile=Initialiser(19);
int nombreDepilage;
Empilage(pile,4);
Empilage(pile,6);
Empilage(pile,8);
Empilage(pile,10);

nombreDepilage=Depilage(pile);
Affichage(pile);
printf("\n \n \n L'element depile : %d \n \n  ",nombreDepilage);
return 0;
}
```

***-La compilation :***

```
8

6

4

19

NULL


L'element depile : 10
```



Scanned with CamScanner

# Les files :

## 7)-Créer la structure Element :

*Comme pour les piles, chaque élément de la file sera de type Element.*

```
struct Element{
    int valeur;
    Element *suivant;
};
```

## 8)-Créer la structure Pile :

*Nous disposerons toujours du premier élément et nous pourrons remonter jusqu'au dernier.*

```
struct File{
    Element *premier;
};
```

```
File *Initialiser(int valeur){
    Element *nouveau=(Element*)malloc(sizeof(Element));
    File *file=(File*)malloc(sizeof(File));
    if(nouveau==NULL || file==NULL){
        exit(EXIT_FAILURE);
    }
    nouveau->valeur=valeur;
    nouveau->suivant=NULL;

    file->premier=nouveau;
}
```

## 9)-La fonction d'emfilage :

*La fonction qui ajoute un élément à la file est appelée fonction « d'enfilage ».*



Scanned with CamScanner

```c
void Enfilage(File *file, int valeur){
 Element *nouveau=(Element*)malloc(sizeof(Element));
 if(file==NULL || nouveau==NULL)    {
    exit(EXIT_FAILURE);
 }
 nouveau->valeur=valeur;
 nouveau->suivant=NULL;
 if(file->premier!=NULL){
    Element *actuel=file->premier;
    while(actuel->suivant!=NULL){
        actuel=actuel->suivant;
     }
     actuel->suivant=nouveau;
 }
 else {
    file->premier=nouveau;
 }
}
```

## 10)-La fonction de défilage :

*Le défilage ressemble étrangement au dépilage. Étant donné qu'on possède un pointeur vers le premier élément de la file, il nous suffit de l'enlever et de renvoyer sa valeur.*

```c
int Defilage(File* file){
 if(file==NULL){
    exit(EXIT_FAILURE);
 }
 Element *adefiler=file->premier;
 int nombreDefile=0;
 file->premier=adefiler->suivant;
 nombreDefile=adefiler->valeur;
 free(adefiler) ;
 return nombreDefile;

}
```



Scanned with CamScanner

## 11)-La fonction d'affichage :

*Il resterait à écrire une fonction afficherFile.*

```
void Affichage(File *file){
    if(file==NULL){
        exit(EXIT_FAILURE);
    }
   Element *actuel=file->premier;
   while(actuel!=NULL){
    printf("%d\t",actuel->valeur);
    actuel=actuel->suivant;
   }
   printf("\tNULL");
}
```

## 12)-La fonction principale :

```
main(){
    File *file;
    int nombreDefilage;
    file=Initialiser(23);
    Enfilage(file, 1);
    Enfilage(file, 2);
    Enfilage(file, 3);
    Enfilage(file, 4);
    Enfilage(file, 9);
    Enfilage(file, 11);

    nombreDefilage=Defilage(file);
    Affichage(file);
    printf("\n \n \n Le nombre defile est : %d \n \n  ",nombreDefilage);
return 0;
}
```



Scanned with CamScanner

***-La compilation :***

## ***CONCLUSION :***

*Ce TP nous a donné une opportunité énorme pour renforcer notre niveau dans les listes chaînées soit simple ou double et d'être apte de maitriser les pointeurs.*

*En résumé, les piles et les files permettent d'organiser en mémoire des données qui arrivent au fur et à mesure.*



Scanned with CamScanner

UNIVESITE SIDI MOHAMED BEN ABDELLAH

Ecole Nationale des Sciences Appliquées

# STRUCURES DES DONNEES EN C

## TP6 : Les arbres


- **Préparé par :** -SELLAK MOUAD - N°207
  -RHARZOZ SOUFIANE - N°198
- **Encadré par** : - Pr. HRAOUI SAID


- **Année universitaire** : 2019/2020



Scanned with CamScanner

## INTRODUCTION :

*Les arbres sont des structures des données très importantes dans l'organisation des données dans la mémoire, ainsi qu'ils jouent un rôle énorme dans la recherche d'une donnée déterminée, surtout les arbres binaires de recherches qui suivent un certain nombre de principes différemment des arbres généraux.*

*Grâce à ce TP on découvrir plus d'avantages et astuces concernant les arbres binaires.*

## OBJECTIFS DU TP :

*Le but de ce TP consiste à maitriser bien la notion des arbres et saisir leurs performances et utilités.*

## DEVELOPPEMEMT :

*Voici les prototypes du TP tout entier :*

```
// Definir les prototypes:

typedef struct Arbre arbre;
arbre* Initialiser(int valeur);
void Inserer(arbre **noued, int newvalue);
void Ajouter(arbre* A, int V);
void Afficher_Pre(arbre *Arbre);
void Afficher_Sym(arbre *Arbre);
void Afficher_Pos(arbre *Arbre);
arbre* SuppNd(arbre *A, int V);
int max(int a, int b);
int Hauteur(arbre* A);
int Nbrnoeuds(arbre *arbre);
```



Scanned with CamScanner

*La structure d'arbre est similaire à celle des listes doublement chainées.*

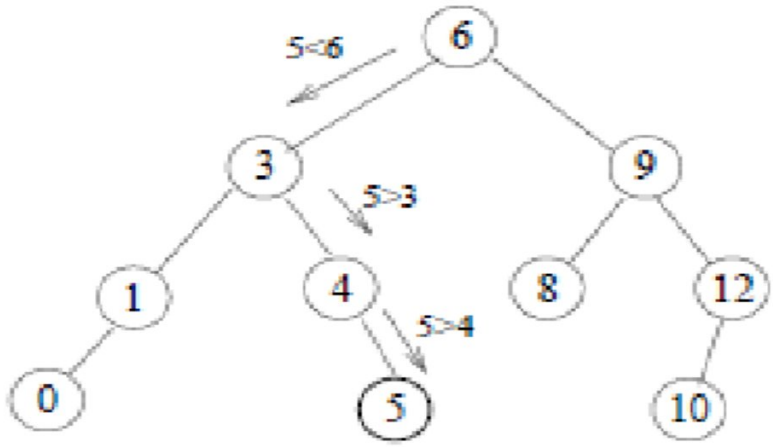
```
// 1)- Définir la strucure :arbre
struct Arbre{
    int valeur;
    arbre *fgauche;
    arbre *fdroit;
};
```

*L'initialisation d'un arbre se fait en même principe que les autres structures des données.*

```
// 2)- Initialisation de l'arbre:
arbre* Initialiser(int valeur){
    arbre* NA=(arbre*)malloc(sizeof(arbre));
    NA->valeur=valeur;
    NA->fdroit=NULL;
    NA->fgauche=NULL;
}
```

*L'insertion d'un élément dans un arbre binaire ne se fait pas n'importe quoi, mais il suit un algorithme plus précis,*

*Découvrons le alors !*






Scanned with CamScanner

```c
// 3)- Inserer un élémént dans l'arbre:
        // Méthode 1:
 void Inserer(arbre **noued, int newvalue){
    if((*noued)==NULL){
        *noued=(arbre*)malloc(sizeof(arbre));
        (*noued)->valeur=newvalue;
        (*noued)->fdroit=NULL;
        (*noued)->fgauche=NULL;
    }
    else{
        if(newvalue>(*noued)->valeur)
         Inserer(&(*noued)->fdroit,newvalue);
        else
         Inserer(&(*noued)->fgauche,newvalue);
    }
}
     // Méthode 2:
void Ajouter(arbre* A, int V){
    arbre* P=(arbre*)malloc(sizeof(arbre));
    P->valeur=V;
    P->fdroit=NULL;
    P->fgauche=NULL;
    if(A==NULL)
    A=P;
    else{
        if(V<=A->valeur){
            if(A->fgauche==NULL)
            A->fgauche=P;
            else
            Ajouter(A->fgauche,V);
        }
        else{
        if(A->fdroit==NULL)
            A->fdroit=P;
            else
            Ajouter(A->fdroit,V);
        }
    }
}
```



Scanned with CamScanner

*Pour parcourir et afficher un arbre il existe plusieurs méthodes à faire :*

```
// 4)- Affichage de l'arbre:
        // Préordre: Examiner la racine puis les sous arbres.
void Afficher_Pre(arbre *Arbre){
    if(Arbre!=NULL){
        printf(" %d ", Arbre->valeur);
        Afficher_Pre(Arbre->fgauche);
        Afficher_Pre(Arbre->fdroit);
    }
}
        // symétrique : Afficher dans l'ordre croissant
void Afficher_Sym(arbre *Arbre){
    if(Arbre!=NULL){
        Afficher_Sym(Arbre->fgauche);
        printf(" %d ", Arbre->valeur);
        Afficher_Sym(Arbre->fdroit);
    }
}
        // Postordre : Examiner les sous arbres puis la racine.
void Afficher_Pos(arbre *Arbre){
    if(Arbre!=NULL){
        Afficher_Pos(Arbre->fgauche);
        Afficher_Pos(Arbre->fdroit);
        printf(" %d ", Arbre->valeur);
    }
}
```

*L'algorithme permettant de supprimer un nœud d'un arbre est le suivant :*

*-Si le nœud à supprimer n'a pas de fils, il suffit de le supprimer.*

*-Si le nœud à supprimer a un seul fils, on le remplace par son fils et on supprime ce dernier.*

*-Si le nœud à supprimer a deux fils, on le remplace par l'extrémité du bord gauche du sous-arbre droit, généralement on le remplace par l'élément le plus petit dans son sous arbre droit.*



Scanned with CamScanner

```c
// 5)-Supprimer un noeud:
arbre* SuppNd(arbre *A, int V){
    arbre *N=A, *NN;  // N=noeud  NN=nouveau_noeud
    arbre **P=&A, **NP; // P=père   NP=nouveau_pere
    while(N!=NULL){  // Recherche de noeud a supprimer:
        if(V==N->valeur)
        break;
        if(V<N->valeur){
            P=&N->fgauche;
            N=N->fgauche;
        }
        else{
            P=&N->fdroit;
            N=N->fdroit;
        }
    }
    if(N->fgauche==NULL){
        if(N->fdroit==NULL){ // feuille
            *P=NULL;
            free(N);
        }
        else{
            *P=N->fdroit;  // noeud à fils droit
            free(N);
        }
    }
    else{
        if(N->fdroit==NULL){ // noeud à fils gauche
            *P=N->fgauche;
            free(N);
        }
        else{     // noeud a fils droit et fils gauche
            NN=N->fdroit;
            NP=&N->fdroit;
            while(NN!=NULL)
                if(NN->fgauche!=NULL){
                    NP=&N->fgauche;
                    NN=N->fgauche;
                }
                N->valeur=NN->valeur;
                *NP=NN->fdroit;
                free(NN);
        }
    }
    return A;
}
```



Scanned with CamScanner

*Par définition, la hauteur d'un arbre est le nombre de nœuds du chemin le plus long dans l'arbre, on dit aussi profondeur de l'arbre.*

```
// 6)- Routourner la hauteur de l'arbre:

// On aura besoin d'une fonction max:
int max(int a, int b){
    if(a<b)
    return b;
    return a;
}

int Hauteur(arbre* A){
    int h;
    if(A==NULL)
    h=0;
    else
    h=1+max(Hauteur(A->fdroit),Hauteur(A->fgauche));
    return h;
}
```

```
// 7)-Nombres des noeuds

int Nbrnoeuds(arbre *arbre){
    int n;
    if(arbre==NULL)
        n=0;
    else
    n=1+Nbrnoeuds(arbre->fgauche)+Nbrnoeuds(arbre->fdroit);
    return n;
    }
```

*Les arbres binaires et leurs manipulations se basent essentiellement sur la notion de récursivité, cette dernière permet à une telle fonction d'appeler elle-même au cours de s'exécution.*



Scanned with CamScanner

```c
    // La fonction principale
main(){
arbre *A,*B;
A=Initialiser(5);
Ajouter(A,1);
Ajouter(A,2);
Ajouter(A,10);
Inserer(&A,222);
Inserer(&A,770);
Inserer(&A,1000);
printf("\n \n Affichage Preordre (prefixe) : ");
Afficher_Pre(A);
printf("\n \n Affichage symetrique (infixe) : ");
Afficher_Sym(A);
printf("\n \n Affichage postordre (sufixe) : ");
Afficher_Pos(A);
printf("\n \n Apres la suppression : ");
Afficher_Sym(SuppNd(A,1000));
printf("\n \n Nombre de noeudes: %d", Nbrnoeuds(A));
printf("\n \n Hauteur: %d", Hauteur(A));
}
```

## ***Compilation :***

```
Affichage Preordre (prefixe) :  5  1  2  10  222  770  1000

Affichage symetrique (infixe) :  1  2  5  10  222  770  1000

Affichage postordre (sufixe) :  2  1  1000  770  222  10  5

Apres la suppression :  1  2  5  10  222  770

Nombre de noeudes: 6

Hauteur: 4
```



Scanned with CamScanner

## ***CONCLUSION :***

*Force est d'avancer qu'on a appris tant des astuces concernant les arbres binaires et leur rapidité dans la manipulation des données, d'une autre part la notion de la récursivité trouve sa place dans la résolution des problèmes des arbres.*

*Bref, les arbres binaires dans la programmation jouent le rôle des arbres dans la nature.*



Scanned with CamScanner

UNIVESITE SIDI MOHAMED BEN ABDELLAH

Ecole Nationale des Sciences Appliquées

# STRUCURES DES DONNEES EN C

## TP7 : Les Tables d'hachages


- **Préparé par :** -SELLAK MOUAD - N°207
  -RHARZOZ SOUFIANE - N°198
- **Encadré par** : - Pr. HRAOUI SAID


- **Année universitaire** : 2019/2020



Scanned with CamScanner

## INTRODUCTION :

*Comme on l'a vu précédemment, les listes chainées sont plus flexibles que les tableaux, car elles permettent d'insérer et éliminer des élément et d'autre opérations, alors qu'il est difficile de faire ça pour les tableaux, d'autre part, la recherche d'un élément est un peu dur pour les listes chainées, car les éléments ne sont pas indexés, si par exemple l'élément voulu se trouve juste avant le dernier, alors il faut parcourir la liste tout entière pour y arriver, autrement pour les tableaux il suffit de noter l'indice d'élément à récupérer.*

*Les tables d'hachages forment une combinaison entre les tableaux et les listes chainées, ils permettent à la fois d'ajouter, retirer et récupérer un élément.*

## OBJECTIFS DU TP :

*-Découvrir la notion des tables d'hachage.*

*-Confirmer les défauts des listes chainées ainsi que les tableaux.*

*-Saisir les performances des tables d'hachages.*

## DEVELOPPEMEMT :

*D'abord on déclare notre structure Elève contenant une chaine de caractère qui va porter le nom de l'élève et un pointeur vers l'élève suivant, puis on déclare la table d'hachage qu'on va utiliser plus tard.*



Scanned with CamScanner

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define size 20

// 1)-La structure Eleve:

typedef struct Eleve Eleve;
struct Eleve{
    char nom[10];
    Eleve *suivant;
};

// La déclaration du table d'achage:
Eleve *TH[size];
```

*La fonction d'hachage est très importante pour transformer la clé (chaine de caractère) en un indice entier compris entre 1 et la taille de la table.*

```c
// 2)- La fonction d'hachage:

int Hash(char *nom){
    int i,H=0;
    for(i=0;i<strlen(nom);i++){
        H+=nom[i];
    }
    return H%size;
}
```

*Avant d'insérer des éléments dans la table, on l'initialise d'abord à NULL.*

```c
// 3)-Initialisation de la Table d'hachage:
void initialiser(){
    int i;
    for(i=0;i<size;i++){
        TH[i]=NULL;
    }
}
```



Scanned with CamScanner

*Dans le remplissage de la tale on doit prendre en considération le cas des collisions.*

```
// 4)-Remplissage de la Table d'hachage:
  // Methode 1:
void Insertion(Eleve *E){
/* Pour gérer les collisions on va utiliser la méthode
du chainnage,s'il y a deux noms qui ont le même indice
renvoyé par la fonction d'hachage on va les stocker dans
une liste chainée. */
    if(E==NULL)
    exit(1);
    int ix=Hash(E->nom);
    E->suivant=TH[ix];
    TH[ix]=E;
}
```

*Dans cette deuxième fonction, le type de retour est false, ça veut dire qu'on ajoute un élément juste dans une case vide.*

```
// Methode 2:
bool Ajout(Eleve *E){
/*On peut également utiliser la méthode d'adressage ouvert,
on va parcourir le tableau et s'il y a une case vide on va
la remplir par l'élément en question.
*/  if(E==NULL) return false;
    int i,j=0,ix=Hash(E->nom);
    for(i=0;i<size;i++){
     j=(i+ix)%size;
    if(TH[j]==NULL){
    TH[j]=E;
    return true;
    }
 }
return false;
}
```



Scanned with CamScanner

```c
// 5)- Affichage de la Table d'hachage:

void Afficher(){
    int i;
    printf("\n \t Indice\t |Eleves \t \n \n");
    for(i=0;i<size;i++){
        if(TH[i]==NULL){
            printf("\t %d \t----\n",i);
        }
        else{
            printf("\t %d \t ",Hash(TH[i]->nom));
            Eleve *tmp=TH[i];
            while(tmp!=NULL){
                printf("%s- ",tmp->nom);
                tmp=tmp->suivant;
            }
            printf("\n");
        }
    }
}
```

***La fonction main :***

```c
main(){
Eleve *TH[size];
initialiser();
Eleve E={"mouad"};
Eleve F={"mouda"};
Eleve G={"sellak"};
Eleve H={"fatima"};
Eleve I={"karima"};
Eleve J={"marika"};
Eleve K={"Khalid"};
Eleve L={"esllak"};
Eleve M={"kalles"};
Eleve N={"mohamed"};
```



Scanned with CamScanner

*Afin de voir le problème des collisions qui se pose pour les tables d'hachages, on va afficher les indices retournés par la fonction d'hachage.*

```
printf(" \n %s --> %d ",E.nom,Hash(E.nom));
printf(" \n %s --> %d ",F.nom,Hash(F.nom));
printf(" \n %s --> %d ",G.nom,Hash(G.nom));
printf(" \n %s --> %d ",H.nom,Hash(H.nom));
printf(" \n %s --> %d ",I.nom,Hash(I.nom));
printf(" \n %s --> %d ",J.nom,Hash(J.nom));
printf(" \n %s --> %d ",K.nom,Hash(K.nom));
printf(" \n %s --> %d ",L.nom,Hash(L.nom));
printf(" \n %s --> %d ",M.nom,Hash(M.nom));
printf(" \n %s --> %d \n \n ",N.nom,Hash(N.nom));
```

```
mouad --> 14
mouda --> 14
sellak --> 16
fatima --> 6
karima --> 9
marika --> 9
Khalid --> 9
esllak --> 16
kalles --> 16
mohamed --> 11
```

*Donc, il parait bien qu'il y a des noms qui ont le même indice, ce problème revient à la probabilité que les lettres de deux noms différents sont les mêmes, malgré qu'ils sont mal triés, comme le cas de (mouad) et (mouda), et malheureusement la fonction d'hachage ne considère pas l'ordre des lettres, elle calcule juste la somme de leurs équivalences numériques (ASCII), mais heureusement ce problème va être résout par les deux méthodes d'insertion mentionnées ci-dessus, et voila l'affichage ci-dessous.*



Scanned with CamScanner

## L'ajout avec adressage ouvert :

```
Ajout(&E);
Ajout(&E);
Ajout(&G);
Ajout(&H);
Ajout(&I);
Ajout(&J);
Ajout(&K);
Ajout(&L);
Ajout(&M);
Ajout(&N);
Afficher();
```

```
Indice  |Eleves

0       ----
1       ----
2       ----
3       ----
4       ----
5       ----
6        fatima-
7       ----
8       ----
9        karima-
9        marika-
9        Khalid-
11       mohamed-
13      ----
14       mouad-
14       mouda-
16       sellak-
16       esllak-
16       kalles-
19      ----
```

## L'ajout avec le chainage :

```
Insertion(&E);
Insertion(&F);
Insertion(&G);
Insertion(&H);
Insertion(&I);
Insertion(&J);
Insertion(&K);
Insertion(&L);
Insertion(&M);
Insertion(&N);
Afficher();

}
```

```
Indice  |Eleves

0       ----
1       ----
2       ----
3       ----
4       ----
5       ----
6        fatima-
7       ----
8       ----
9        Khalid- marika- karima-
10      ----
11       mohamed-
12      ----
13      ----
14       mouda- mouad-
15      ----
16       kalles- esllak- sellak-
17      ----
18      ----
19      ----
```



Scanned with CamScanner

## ***CONCLUSION :***

*En guise de conclusion, les tables d'hachages représentent à la fois des tableaux et des listes chainées ; Ils représentent les tableaux grâce à l'indexage donnée par la fonction d'hachage, et ils représentent des listes chainées en chainant les éléments de même indice afin de résoudre le problème des collisions.*

*Il est bien de noter que plus la fonction d'hachage est performante, plus que les collisions se réduisent.*

***FIN…***



Scanned with CamScanner

UNIVESITE SIDI MOHAMED BEN ABDELLAH

Ecole Nationale des Sciences Appliquées

# STRUCURES DES DONNEES EN C

## TP8 : Les algorithmes du tri


- **Préparé par :** -SELLAK MOUAD - N°207
  -RHARZOZ SOUFIANE - N°198
- **Encadré par** : - Pr. HRAOUI SAID


- **Année universitaire** : 2019/2020



Scanned with CamScanner

## INTRODUCTION :

*Le problème qui se pose souvent pour les structures des données est de chercher une méthode pour les trier, pour accéder à une donnée précise rapidement. Comme on l'a vu précédemment pour les tableaux, le principe de tri reste le même, mais ce qui va changer est qu'on va s'intéresser maintenant à des listes simplement chainées au lieu des tableaux.*

## OBJECTIFS DU TP :

*Le but de ce TP est de découvrir des méthodes efficaces pour trier des listes simplement chainées, notamment maitriser le tri par insertion, sélection, bulles, fusion.*

## DEVELOPPEMEMT :

*Avant d'entamer les différents types de tri, on préfère de citer toutes les fonctions qu'on en aura besoin :*

*Listons d'abord tous les prototypes de toutes les fonctions qu'on va utiliser pour le tri : par insertion, bulles, sélection, et pour le tri à fusion, vu sa complexité on l'a fait dans un programme tout seul.*



Scanned with CamScanner

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

// Les prototypes:

typedef struct Element Element;
typedef struct Pile Pile;
typedef struct Liste Liste;
Liste* Allouer();
void Empiler(Pile *, int );
void AfficherP(Pile *);
void AjouDebut(Liste* , int );
void AfficherL(Liste*);
void Permuter(Element *, Element *);
void SuppVal(Liste *, int );
void SuppDebut(Liste *);
Element *Precedent(Element*, Liste*);

void Tri_Selection(Liste* );
void Tri_Bulles(Liste* );
void Tri_Insertion(Liste *);
```

```c
// La strucure Element
struct Element{
    int valeur;
    Element *suivant;
};

// La strucure Pile
struct Pile{
    Element *premier;
};


// La structure Liste
struct Liste{
    int longueur;
    Element *Tete;
};
```



Scanned with CamScanner

```c
// L'allocation de la liste
Liste* Allouer(){
    Liste* New=(Liste*)malloc(sizeof(Liste));
if(New==NULL){
    exit(EXIT_FAILURE);
}
    New->longueur=0;
    New->Tete=NULL;

    return New;
}

// Empiler un element dans la pile
void Empiler(Pile *pile, int valeur){
    Element *New=(Element*)malloc(sizeof(Element));
    if(pile==NULL || New==NULL){
        exit(EXIT_FAILURE);
    }
    New->valeur=valeur;
    New->suivant=pile->premier;
    pile->premier=New;
}

// Inserer un element au début de la liste
void AjouDebut(Liste* liste, int valeur){
    Element *New=(Element*)malloc(sizeof(Element));
    if(liste==NULL || New==NULL){
        exit(EXIT_FAILURE);
    }
    New->valeur=valeur;
    New->suivant=liste->Tete;
    liste->Tete=New;
    liste->longueur++;
}
```



Scanned with CamScanner

```c
// Afficher la liste
void AfficherL(Liste* liste){
    Element* tmp=liste->Tete;
    printf("\n ");
    while(tmp!=NULL){
        printf(" %d -> ", tmp->valeur);
        tmp=tmp->suivant;
    }
    printf(" NULL \n ");
}

// Afficher la pile
void AfficherP(Pile *pile){
    Element* tmp=pile->premier;
    while(tmp!=NULL){
        printf("\t\t%d\n", tmp->valeur);
        tmp=tmp->suivant;
    }
    printf("\t\tNULL\n\n");

}
```

*Pour les fonctions d'insérer et afficher, on peut faire une seule fonction pour les piles et les listes chainées, mais pour plus d'organisation on a les fait séparément.*

```c
// Permuter deux elements
void Permuter(Element *a, Element *b){
    int t;
    t=a->valeur;
    a->valeur=b->valeur;
    b->valeur=t;
}
```



Scanned with CamScanner

```c
// Retourner le precedent d'un element
Element *Precedent(Element* E, Liste *L){
    Element *R;
    if(E==(L->Tete))
    return NULL;
    else
    for(R=L->Tete; R->suivant!=E; R=R->suivant);
    return R;
}
```

*Cette fonction ci-dessus, on a la fait pour y utiliser à supprimer un élément de la liste simplement chainée, puisque cette dernière ne contient pas un pointeur vers l'élément précédent d'un élément donné, comme le cas des listes doublement chainées.*

```c
// Supprimer un element par valeur
void SuppVal(Liste *liste, int Valeur){
    Element* tmp=liste->Tete;
    while(tmp!=NULL){
        if(tmp->valeur==Valeur)
        Precedent(tmp,liste)->suivant = tmp->suivant;
        tmp=tmp->suivant;
    }
    liste->longueur--;
}

// Supprimer la tete d'une liste
void SuppDebut(Liste* liste){
    Element* tmp=liste->Tete;
    liste->Tete=tmp->suivant;
    free(tmp);
}
```

*On a ajouté la fonction de suppression au début, puisque celle de suppression par valeur ne peut pas supprimer le*



Scanned with CamScanner

*premier élément car ce dernier n'a pas d'élément précédent.*

## ➔ ***Le tri par sélection :***

### Tri par sélection

- Le tri par sélection consiste simplement à sélectionner l'élément le plus petit de la suite à trier, à l'enlever, et à répéter itérativement le processus tant qu'il reste des éléments dans la suite.
- Au fur et à mesure les éléments enlevés sont stockés dans une pile.
- Lorsque la suite à trier est stockée dans un tableau on s'arrange pour représenter la pile dans le même tableau que la suite : la pile est représentée au début du tableau, et chaque fois qu'un élément est enlevé de la suite il est remplacé par le premier élément qui apparaît à la suite de la pile, et prends sa place.
- Lorsque le processus s'arrête la pile contient tous les éléments de la suite triés dans l'ordre croissant.

*Le principe de ce tri est comme suit : on cherche l'élément le plus petit de la liste, puis en le sélectionne et l'insérer dans une pile initialement vide, puis on supprime cet élément de la liste, et on répète le processus tant que la longueur de la liste n'est pas nulle.*



Scanned with CamScanner

```c
/* Le tri par selection : chercher le petit élément de la liste
 et l'empiler dans la pile.*/
void Tri_Selection(Liste* liste){
Pile* pile=(Pile*)malloc(sizeof(Pile)); // Création de la pile.
pile->premier=NULL;
Element*tmp=liste->Tete;
int min;
while(liste->longueur >= 0 ){
    tmp=liste->Tete;
    min=tmp->valeur;
    while(tmp!=NULL){  // Chercher le min
     if(min>tmp->valeur)
        min=tmp->valeur;
        tmp=tmp->suivant;
  }
  Empiler(pile, min);
  AfficherP(pile);
  if(min==liste->Tete->valeur)
  SuppDebut(liste);
  else
  SuppVal(liste,min);
  /* Donc la longueur va dimnuer et reboucler
  jusqu' elle soit nulle.*/
 }
}
```

➔ ***Le tri à bulles :***

## Tri à bulle

- Le tri à bulle consiste à parcourir le tableau, tant qu'il n'est pas trié, et à permuter les couples d'éléments consécutifs mal ordonnés.
- On sait que le tableau est trié si lors d'un parcours, aucun couple d'éléments n'a été permuté.

*Comme il est mentionné dans le cours ci-dessus le principe de ce type de tri est clair.*



Scanned with CamScanner

```c
/* Le tri a bulles: comparer les paires consécutifs et les pérmuter
 jusqu'a ce que tous les paires sont triés:*/
void Tri_Bulles(Liste* liste){
    Element *tmp,*cmp;
    for(tmp=liste->Tete; tmp->suivant!=NULL; tmp=tmp->suivant){
        for(cmp=tmp; cmp!=NULL; cmp=cmp->suivant){
            if(tmp->valeur > cmp->valeur)
            Permuter(tmp,cmp); // Appeler la fonction Permuter
        }
    }AfficherL(liste);
    /*  Autre méthode:
    Element* tmp;
    int ech=1;
    while(ech>0){
        ech=0;
        tmp=liste->Tete;
        while (tmp!=NULL){
            if(tmp->valeur > tmp->suivant->valeur){
            Permuter(tmp,tmp->suivant);
            ech++;
            }
            tmp=tmp->suivant;
        }
    }*/
}
```

## ➔ ***Le tri par insertion :***

# Tri par insertion

- Le tri par insertion consiste à insérer les éléments de la suite les uns après les autres dans une suite triée initialement vide.
- Lorsque la suite est stockée dans un tableau la suite triée en construction est stockée au début du tableau.
- Lorsque la suite est représentée par une liste chainée on insère les maillons les uns après les autres dans une nouvelle liste initialement vide.



Scanned with CamScanner

*On remarque que d'après le cours, ce type de tri est ressemble un petit peu au celui de sélection.*

```
/* Tri par insertion: inserer les éléments triés les uns apres les autres
 dans une liste initilement vide:*/
void Tri_Insertion(Liste *liste){
    Liste *Nliste=Allouer(); Element* tmp; int max;
 while(liste->longueur >= 0){
    tmp=liste->Tete;
    max=tmp->valeur;
    while(tmp!=NULL){
        if(max < tmp->valeur)
            max=tmp->valeur;
            tmp=tmp->suivant; }
AjouDebut(Nliste, max); // Appel de la fonction Ajout au debut.
AfficherL(Nliste); //On affiche à chaque iteration pour visualiser le principe
if(max==liste->Tete->valeur)
SuppDebut(liste);
else SuppVal(liste, max);
  } /*  Autre méthode:
Element *tmp,*cmp=NULL,*help=liste->Tete;
for(tmp=liste->Tete; tmp!=NULL; tmp=tmp->suivant){
    cmp=tmp;
    while( (cmp->valeur < tmp->valeur) && cmp!=help){
        Permuter(cmp,Precedent(cmp,liste));
        cmp=Precedent(cmp,liste);
    } }*/
}
```

## ➔ ***La fonction main :***



Scanned with CamScanner

```c
main(){

Liste *liste=Allouer();
int i,n,ele,choix;
printf("\n Combien d'element vous voulez inserer dans votre liste: ");
scanf("%d", &n);
printf("Entrez vos elements: \n ");
for(i=1;i<=n;i++){
    scanf("%d",&ele);
    AjouDebut(liste, ele);
}

printf("\n Avant le tri:    \n    ");
     AfficherL(liste);

 printf("\n\n  ---------------------- Menu ----------------------\n \n ");
 printf("\n 1. Le tri par selection: \n ");
 printf("\n 2. Le tri a bulles: \n ");
 printf("\n 3. Le tri par insertion: \n ");

 printf("\n \n Entrez votre choix: ");
 scanf("%d",&choix);

 switch(choix){

    case 1 :
            printf("\n Apres le tri:\n ");
            Tri_Selection(liste);
            break;

    case 2 :
            printf("\n Apres le tri:\n ");
            Tri_Bulles(liste);
            break;

    case 3 :
            printf("\n Apres le tri:\n ");
            Tri_Insertion(liste);
            break;
 }

 return 0;
 }
```



Scanned with CamScanner

## ➔ *La compilation :*

```
 Combien d'element vous voulez inserer dans votre liste: 6
Entrez vos elements:
 1
8
3
99
34
6

 Avant le tri:

  6 ->  34 ->  99 ->  3 ->  8 ->  1 ->  NULL


  ---------------------- Menu ----------------------


 1. Le tri par selection:

 2. Le tri a bulles:

 3. Le tri par insertion:
```

```
Entrez votre choix: 1

Apres le tri:
              1
              NULL

              3
              1
              NULL

              6
              3
              1
              NULL

              8
              6
              3
              1
              NULL

              34
              8
              6
              3
              1
              NULL

              99
              34
              8
              6
              3
              1
              NULL
```



Scanned with CamScanner

```
 Combien d'element vous voulez inserer dans votre liste: 7
Entrez vos elements:
 2
8
5
99
2
1
4

 Avant le tri:

  4 ->  1 ->  2 ->  99 ->  5 ->  8 ->  2 ->  NULL


  ---------------------- Menu ----------------------


 1. Le tri par selection:

 2. Le tri a bulles:

 3. Le tri par insertion:


 Entrez votre choix: 2

 Apres le tri:

  1 ->  2 ->  2 ->  4 ->  5 ->  8 ->  99 ->  NULL
```

```
 Combien d'element vous voulez inserer dans votre liste: 6
Entrez vos elements:
 11
6
77
25
1
67

 Avant le tri:

  67 ->  1 ->  25 ->  77 ->  6 ->  11 ->  NULL


  ---------------------- Menu ----------------------


 1. Le tri par selection:

 2. Le tri a bulles:

 3. Le tri par insertion:


 Entrez votre choix: 3

 Apres le tri:

  77 ->  NULL

  67 ->  77 ->  NULL

  25 ->  67 ->  77 ->  NULL

  11 ->  25 ->  67 ->  77 ->  NULL

  6 ->  11 ->  25 ->  67 ->  77 ->  NULL

  1 ->  6 ->  11 ->  25 ->  67 ->  77 ->  NULL
```



Scanned with CamScanner

## ➔ Le tri par fusion :

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

/* Le tri par fusion :suit le principe Diviser pour régner,on divise
 la liste à des paires récursivement, puis on les fusione triés.*/

// Les prototypes:

typedef struct Liste  *   Element;
void InsererFin(Element *, int );
void Afficher(Element );
void Diviser(Element , Element * , Element * );
void Tri_Fusion(Element *  );
Element Fusioner(Element , Element );

struct Liste{
    int valeur;
    Element suivant;
};

Element TeteListe;// C'est une variable globale représentant notre liste.
```

```c
void InsererFin(Element *DernierElem, int valeur){
Element NewElem=(Element)malloc(sizeof(struct Liste));
NewElem->valeur=valeur; // Remplir l'élément à insérer
NewElem->suivant=NULL;
if(*DernierElem==NULL){  // Le cas d'insertion au début
*DernierElem=NewElem;
TeteListe=*DernierElem;
}
else{
    (*DernierElem)->suivant=NewElem; // Inserer a la fin
    *DernierElem=NewElem;// Le nouveau element devient le dernier
  }
}

void Afficher(Element Tete){
Element Actuel=Tete; // Initialiser l'actuel au head de la liste
    while(Actuel!=NULL){
        printf(" %d -> ", Actuel->valeur);
        Actuel=Actuel->suivant;
    }
    printf("NULL\n");
}
```



Scanned with CamScanner

```c
void Diviser(Element liste_tete, Element * liste_gauche, Element * liste_droite){
/* Cette fonction sert à diviser notre liste à deux moitiés: on va créer deux éléments
qui vont parcourir la liste jusqu'ariver à sa moitié et donc retourner deux listes
représentant les deux moitiées */
Element rapide, lent;

if(liste_tete==NULL || liste_tete->suivant==NULL){
    *liste_gauche=liste_tete;
    *liste_droite=NULL;
}else
lent=liste_tete;
rapide=liste_tete->suivant; // Maintenant le lent et le rapide sont consécutifs
while(rapide!=NULL){
    rapide=rapide->suivant;
    if(rapide!=NULL){
        lent=lent->suivant;
        rapide=rapide->suivant;
    }/* Cette boucle While permet de translater le rapide deux fois que le lent et donc
    itérativement jusqu'à le lent situe au milieu de la liste, et la rapide devient NULL */
  } // maintenant le lent se situe au milieu de la liste
 *liste_gauche=liste_tete;
 *liste_droite=lent->suivant;
lent->suivant=NULL; // cette instruction est imporatante pour séparer les deux listes
printf("\n \n \t"); Afficher(*liste_gauche);printf(" \t ");  Afficher(*liste_droite);
}
Element Fusioner(Element E_gauche, Element E_droit){
// Cette fonction permettra de fusionner recursivement les éléments en les trier
    Element Liste_Fusione=NULL; // La liste à retrourner
    if(E_gauche==NULL) return E_droit;
    else if(E_droit==NULL) return E_gauche;
    if(E_gauche->valeur < E_droit->valeur){
    Liste_Fusione=E_gauche;
    Liste_Fusione->suivant=Fusioner(E_gauche->suivant, E_droit); }
 else{
    Liste_Fusione=E_droit;
    Liste_Fusione->suivant=Fusioner(E_gauche, E_droit->suivant); }
return Liste_Fusione; // Maintenant la liste est trie
}

void Tri_Fusion(Element *  ListeFinale){
Element tete_liste=*ListeFinale;
Element Liste_gauche=NULL, Liste_droite=NULL;
if(tete_liste==NULL || tete_liste->suivant== NULL) return; /* Ce test est tres important
pour la recursivité de la fonctio, c'est celui-ci qui va arrêter le procéssus.*/
Diviser(tete_liste,&Liste_gauche,&Liste_droite); // D'abord on divise notre liste
Tri_Fusion(&Liste_gauche); // On tri la liste gauche toute seule
Tri_Fusion(&Liste_droite); // On tri la liste droite toute seule
*ListeFinale=Fusioner(Liste_gauche,Liste_droite);   // On fusionne les deux listes.
}
```



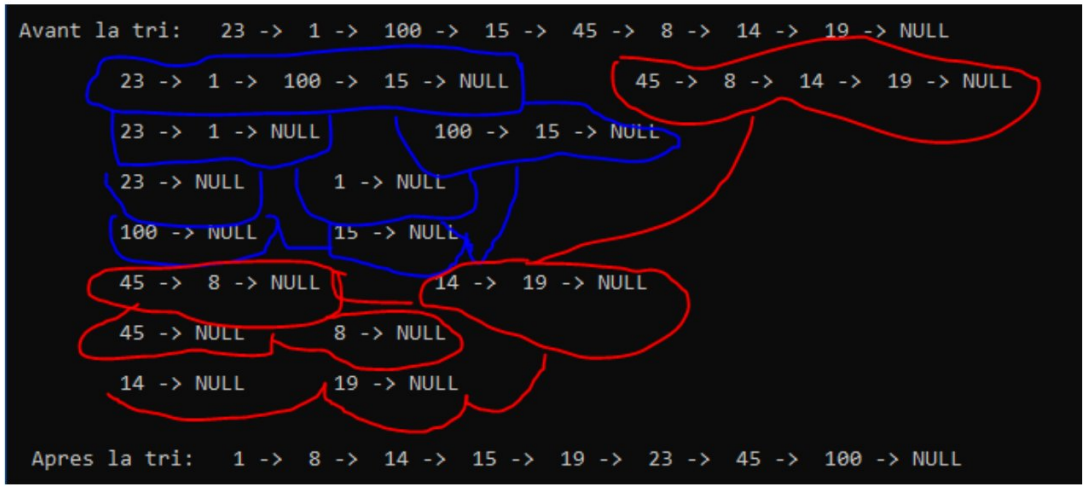
Scanned with CamScanner

## ➔ ***La fonction main :***

```
main(){
Element tete=NULL;
InsererFin(&tete,   23);
InsererFin(&tete,   1);
InsererFin(&tete, 100);
InsererFin(&tete,   15);
InsererFin(&tete,   45);
InsererFin(&tete,   8);
InsererFin(&tete, 14);
InsererFin(&tete,   19);

printf("\n Avant la tri:  ");
Afficher(TeteListe);
Tri_Fusion(&TeteListe);
printf("\n \n \n  Apres la tri:  ");
Afficher(TeteListe);

return 0;
}
```




*On remarque que la liste est divisée récursivement, puis fusionnée aussi récursivement.*



Scanned with CamScanner

## ***CONCLUSION :***

*En guise de conclusion, les algorithmes de tri sont très importants pour les structures des données, c'est ceux qui les rendent plus flexibles, et facilitent l'accès à un élément donné.*

*On a traité quatre types de tri : le tri par fusion, insertion, sélection, bulles, et chacun d'eux a sa propre complexité,*

*En plus il existe d'autres types de tri, notamment le tri par tas, le tri rapide …*

***FIN…***



Scanned with CamScanner